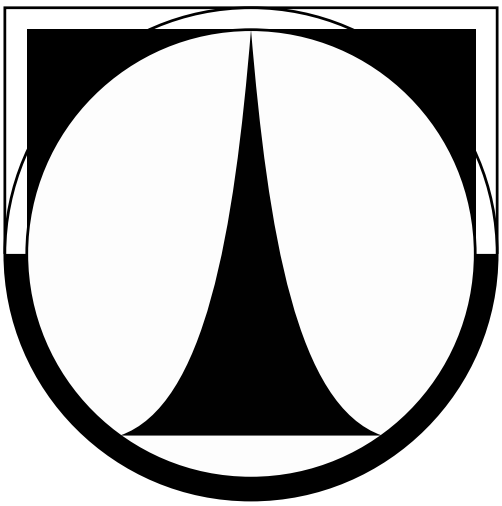
Technická univerzita v Liberci

Fakulta mechatroniky, informatiky a mezioborových studií

BAKALÁŘSKÁ PRÁCE



Ondřej Tůma

Verse modul pro programovací jazyk Python

Ústav nových technologií a aplikované informatiky

Vedoucí bakalářské práce: **Ing. Jiří Hnídek, Ph.D.**

Studijní program: **Informační technologie**

Studijní obor: **Informační technologie**

Liberec 2012

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 Informační technologie

Studijní obor: Informační technologie

Verse modul pro programovací jazyk Python
Verse module for Python programming language

Bakalářská práce

Autor: **Ondřej Tůma**

Vedoucí práce: Ing. Jiří Hnídek, Ph.D.

V Liberci 1.4.2012

Poděkování

Na tomto místě bych chtěl poděkovat zejména svému vedoucímu bakalářské práce Ing. Jiřímu Hnídkovi, Ph.D. za cenné připomínky a odborné rady, kterými přispěl k vypracování této bakalářské práce. Dále děkuji svojí rodině za trpělivost a klid, který mi pro psaní poskytli.

Použitý software

Tato práce byla vysázena programem LaTeX pod operačním systémem Linux.

Kontakt

E-mail: turbis@vdffree.net

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Anotace

Cílem bakalářské práce „Verse modul pro programovací jazyk Python“ je vytvořit modul pro programovací jazyk Python z knihovny implementující klientskou část síťového protokolu Verse, která je naprogramovaná v programovacím jazyku C.

První část práce tvoří úvod do problematiky tvoření modulů, seznámení se síťovým protokolem Verse a programy k tomu potřebnými. Následuje praktická část bakalářské práce – vytvoření modulu pro Python a vytvoření textového a grafického klienta, na kterých lze demonstrovat funkčnost knihovny.

Klíčová slova

Verse, Python, SWIG, modul, PyGTK, převod, kompilace, Glade

Annotation

The purpose of bachelor's thesis "Verse module for Python programming language"is to create a module for Python programming language, from a library implementing the client-side part of Verse networking protocol, programmed in C language.

The first part contains the introduction to issues with creating modules, a way to get familiar with Verse networking protocol and programs associated with that. The practical part of thesis follows - creation of the Python module, text and visual client, making it possible to demonstrate the library's functionality.

Keywords

Verse, Python, SWIG, modul, PyGTK, convert, compilation, Glade

Obsah

Seznam obrázků

Seznam symbolů a zkratek

API Application Programmers Interface

FPS First Person Shooting

GNOME GNU Object Model Environmen

KDE K Desktop Environment

MMORPG Massively Multiplayer Role Playing Game

TCP Transmision Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

XML Extensible Markup Language

1 Úvod

Téma bakalářské práce „Verse modul pro programovací jazyk Python“ si autor vybral z důvodu velkého zájmu o oblasti programování a počítačových sítí, kterým se nějakou dobu věnuje. Zejména v oblasti počítačové interaktivní grafiky se setkáváme s požadavkem na přenos velkého objemu dat s co nejmenší latencí. Nejznámějším příkladem požadavku je herní aplikace typu Massive(ly)-Multiplayer Online Role-Playing Game (MMORPG) nebo First Person Shooting (FPS). U těchto aplikací se ovšem většinou nesdílí geometrie a topologie objektů, ale sdílejí se pouze polohy jednotlivých avatarů a jejich stavů. Jedním ze síťových protokolů pro tento účel je Verse. Verse je poměrně mladý síťový komunikační protokol umožňující real-time komunikaci mezi grafickými programy, který je v dnešní době implementován, pro příklad, v grafickém programu Blender [12].

Protokol Verse je napsán v programovacím jazyce C. Z toho vyplývají velké nároky na programátora, proto si dal autor za cíl vytvořit jednoduše použitelný modul klientské části knihovny Verse pro programovací jazyk Python. Díky čemuž věří, že se protokol Verse bude více využívat při programování aplikací, kde se jeho vlastnosti hodí.

První část práce tvoří seznámení se síťovým protokolem Verse a úvodem do problematiky tvoření modulů - obsahuje souhrn možných technik pro tvorbu modulu. Ke každé technice je vždy ukázán příklad a vypsány jeho výhody a nevýhody. Následuje praktická část bakalářské práce – vytvoření modulu pro Python, vytvoření textového a grafického klienta s grafickou knihovnou GTK+ [34], rps. PyGTK [21] a GLADE [15].

Textový klient slouží jako ukázka všech implementovaných funkcí a grafický klient demonstruje výhody síťového protokolu Verse.

2 Teoretická část

2.1 Protokol Verse

Vývoj protokolu Verse začal v roce 1999, kdy Eskil Steenberg a Emil Brink napsali první řádky kódu. Vývoj protokolu byl velmi živelný a jeho návrh se častokrát měnil. Konečnou podobu první verze lze nalézt na webových stránkách Verse [30]. Ze způsobu vývoje vzniklo v návrhu několik nedostatků, díky čemuž se skoro nerozšířil.

Verse byl zpočátku financován z fondu Evropské unie v rámci 6. rámcového programu. Po počátečním vývoji Verse protokolu se vývojáři spíše zaměřili na psaní aplikací, které budou protokol Verse využívat a opomněli protokol Verse nadále vyvíjet. Po ukončení financování vývoj kolem protokolu Verse skoro ustal, ačkoliv měl ambice stát se univerzálním síťovým protokolem pro komunikaci mezi grafickými aplikacemi. Mezi nadšenci vzniklo několik forků, mezi nimi i fork pana Jiřího Hnídka. Který ve své disertační práci [5] protokol vylepšuje a přidává mu nové vlastnosti. Bakalářká práce je proto postavena na tomto nově předělaném protokolu Verse. Novou verzi API protokolu Verse lze najít na stránkách Technické univerzity v Liberci [10]. Nové API vychází ze starého API protokolu Verse.

2.1.1 Datový model

Komunikace mezi serverem a různými klienty vychází ze stejného datového modelu. Vlastní uložení dat na straně klienta specifikace protokolu Verse neřeší. Předpokládá se, že data jsou strukturována ve stromové struktuře do uzlů. Každý uzel má svoje ID, vlastníka a svoje přístupová práva. Stromová struktura se skládá z kořenového uzlu s ID=0, který vlastní server a nikdo jiný nemá právo do něj zapisovat. Kořenový uzel musí mít vždy tři potomky:

- uzel s ID = 1 je předek všech uzlů reprezentující avatary - reprezentace uživatele ve virtuální realitě
- uzel s ID = 2 je předek všech uzlů reprezentující uživatele

- uzel s ID = 3 je předek pro kořenové uzly jednotlivých 3D scén

V uzlu s ID=1 vytváří server uzly reprezentující avatary uživatelů, kteří jsou aktuálně přihlášeni. V uzlu s ID=2 server vytvaří uzly reprezentující platné uživatele,tedy i uživatele, kteří nejsou přihlášeni. Poslední potomek slouží ke sdílení dat mezi jednotlivými klienty. Oproti předešlým potomkům má každý uživatel právo kromě čtení i zapisovat.

Uživatel může vytvořit uzly v uzlu, který reprezentuje jeho avatara. Kromě vytvoření uzlu může uživatel vytvořit tagy a skupiny tagů. Jejich svázanost je na principu stromové struktury. Tedy uzel může být rodičem několika skupin tagů a skupina tagů může být jen rodičem tagů. Nelze aby uzel byl rodičem tagu a podobně. Pro upřesnění významu těchto prvků můžeme považovat uzel za objekt, dejme tomu krychli, pak skupina tagů může být například jeho pozice a tagy můžou být jednotlivé souřadnice.

2.1.2 Verse server

Verse server je implementován jako vícevláknová aplikace, která poslouchá na požadavky klientů v samostatném vlákně pomocí TCP [7] socketu. Pro každé nové spojení vytvoří server samostatné vlákno. V tomto vlákně se provede TCP a TLS [6] handshake (sekvence zpráv vyměňovaných mezi dvěma nebo více síťovými zařízeními k zajištění synchronizace přenosů). Po přihlášení uživatele a dohodnutí nového UDP [8] spojení je pro UDP spojení vytvořeno další vlákno tzv. datagramové vlákno. V tomto datagramovém spojení probíhá následná komunikace s klientem.

2.1.3 Verse klient

Bakalářská práce se zabývá implementací klientské části Verse protokolu, dle nového API. Klient komunikuje se serverem pomocí dvou samostatných vláken (datagramové a hlavní), která jsou vytvořena po zavolání funkce *verse_send_connect_request()*. Výměna dat mezi datagramovým a hlavním vláknem se uskutečňuje pomocí funkcí začínajících *verse_send_*. Výčet těchto funkcí lze nalézt v příloze na straně 39. Kvůli předávání dat z datagramového vlákna do vlákna hlavního je nutné, aby klient zaregistroval příslušné

callback funkce, jejich výčet lze taktéž nalézt v příloze na straně 39. Výhoda více vláken spočívá hlavně v tom, že hlavní vlákno může být zaneprázdněno a datagramové vlákno může mezitím komunikovat se serverem. Datagramové vlákno si uchovává přijatá data do té doby, než si o ně hlavní vlákno řekne za pomocí funkce *verse_callback_update()* .

2.2 Techniky vytváření modulu pro programovací jazyk Python

2.2.1 Python

Protokol Verse je napsán v programovacím jazyce C. Tento jazyk je považován za nízkoúrovňový, tedy klade velké požadavky na znalosti programátora. V dnešní době se spíše prosazují objektově orientované programovací jazyky, což jazyk C není.

Python [26] je dynamický objektově orientovaný skriptovací programovací jazyk, který vznikl v roce 1991. Je dostupný na všech běžných platformách (Unix, Windows, Mac OS) a je v posledních letech standartní součástí většiny distribucí systému Linux. Nabízí významnou podporu k integraci s ostatními jazyky a nástroji a přichází s mnoha standardními knihovnami.

Vlastnosti Pythonu:

- dostupnost na všech hlavních platformách
- spolupráce s jinými programovacími jazyky
- výkon - výkonově kritické knihovny jsou implementovány v jazyce C
- objektový jazyk
- interpretovaný jazyk
- lehce čitelný kód
- lehce naučitelný jazyk
- v posledních letech stále více v oblibě mezi programátory

2.2.2 Přepsání zdrojových kódů

Přepsání zdrojového kódu z původní knihovny napsané v jazyce C do programovacího jazyka Python se jistě jeví na první pohled jako nejjednodušší řešení. Problém však nastává při samotné realizaci. Jelikož tato technika je velmi náročná. Jak na znalosti programátora, tak na čas. U velkých projektů může trvat přepsání zdrojových kódů z jednoho programovacího jazyka do jiného programovacího jazyka, dokonce až exponenciálně větší čas než při použití jiné techniky pro vytvoření modulu. Taktéž je potřeba hledět na čas při změnách ve zdrojovém kódu nebo na implementaci nového. Což je velice neefektivní a v mnoha případech i velmi drahé. Další problémy mohou nastat díky rozdílnosti programovacích jazyků - například nepodporované datové typy, ukazatele. Tyto problémy se řeší většinou pracně a zdlouhavě.

Výhoda přepsání zdrojových kódů je v přenositelnosti. Díky vlastnostem Pythonu je možné spustit kód na všech běžných platformách a lze je implementovat do jiných jazyků (java - Jython [35], C# - IronPython [18]). Nevýhodou tohoto řešení je, jak již bylo popsáno výše, pracnost. K vytvoření modulu nebyla tato technika použita z důvodu nedostatků popsaných výše. Mimo jiné klientské části protokolu touto metodou je implementováno již mnoho.

2.2.3 Přepsání pomocí C API

C API je programátorské rozhraní pro Python, které dává programátorům C a C++ přístup k interpretu Pythonu. Toto rozhraní je již obsaženo v samotném Pythonu a sami vývojáři Pythonu doporučují používat tento způsob přepisování kódu. C API má velmi dobře napsanou dokumentaci, která obsahuje mnoho příkladů. Při tvorbě modulu je nutné nejprve vytvořit soubor, který bude obsahovat:

```
#include <Python.h>
```

Tato řádka zapříčiní přidání Python API do projektu. Dále v souboru modulu musí být uvedeny funkce, které chceme volat z Pythonu. Tyto funkce musí být napsány pomocí

Python objektů a slouží k volání funkcí napsaných v jazyku C/C++. Ukázky lze nalézt v referenčním manuálu C API [24]. Následně se musí modul pomocí kompilátoru přeložit.

C API je velmi silný nástroj s velkými možnostmi, ale bohužel tím pádem je i velmi složitý. Díky nutnosti ošetřování vstupů a vyjímek funkcí se kód modulu „nafukuje“ mnoha řádky kódu, které se dost často mohou i opakovat s nepatrnými změnami.

2.2.4 Přepsání pomocí tzv. mezivrstvy

Mezivrstvou se rozumí použití externí knihovny. Knihovna vytvoří za pomoci konfiguračních souborů mezivrstvu, která umožňuje rozšiřování a zapouzdřování jazyků jako jsou: C#, Java, Perl, PHP, Python, Ruby, Tcl a podobné. K vytvoření mezivrstvy existuje několik druhů knihoven. Tyto knihovny používají vlastní pseudojazyky, díky nimž je možné vytvořit jednoduše modul pro Python právě ze zdrojového kódu napsaném v jazyce C. Knihovny z pseudojazyka vygenerují (skoro) vždy kód, který je napsaný C API a následně vytvoří modul pro Python. Knihoven, které se touto problematikou zabývají, je několik. Některé jsou dílem jednoho autora, některé jsou zase dílem mnohdy i několika desítek vývojářů. Předem je nutno říci, že žádná knihovna není naprosto ideální, a proto je nutné zvolit kompromis při výběru.

2.3 Seznámení s programy pro psaní mezivrstvy

2.3.1 SIP

SIP [29] vznikl v roce 1998 a jeho autorem je Phil Thompson, který se na jeho vývoji podílí dodnes. SIP byl původně napsán jako pomocný nástroj k PyQT [22] (Python implementaci QT [27]). Jelikož se autorovi zdál málo využitý, tak SIP zobecnil pro tvorbu mezivstvy mezi C / C++ a Pythonem. SIP je šířen pod licencí Python Software Foundation a je zaštiťován společností RiverBank. Jako jeden z mála se může SIP pyšnit kvalitní dokumentací, která je obohacena velkou škálou příkladů jak pro jazyk C, tak pro jazyk C++. Mimo obvyklých vlastností dokáže SIP zachytávat a zpracovávat vyjímky, které vzniknou v mezivrstvě, což je velmi užitečná vlastnost, zejména při ladění převedeného kódu. Při

tvorbě modulu se musí napřed vytvořit sip soubor a v něm definovat funkce, které chceme používat v Pythonu. Potom musí SIP zaobalit tyto funkce do svého pseudokódu a vznikne soubor s příponou .sbf. Pro další krok je nutné vytvořit konfigurační soubor pro Python (v něm jsou obsažené údaje, jak a co má zkompilovat).

Jednoduchý příklad lze nalézt v příloze na straně 34.

2.3.2 BOOST::PYTHON

Boost::Python [13] je knihovna, která je součástí projektu „Boost C++ libraries“. Projekt vznikl v roce 1998 a jeho autorem je Beman Dawes. Později Dawes sjednotil skupinu lidí, která se podílela na vývoji Boost knihoven. Projekt se může pyšnit kvalitní a velmi rozsáhlou dokumentací, která je doplněna podrobně popsanými příklady.

Knihovna poskytuje veškeré potřebné prostředky k převodu kódu z C++ do Python modulu nebo do Java knihovny. Oproti ostatním knihovnám je BOOST::PYTHON náročnější na paměťové prostředky.

Jednoduchý příklad lze nalézt v příloze na straně 35.

2.3.3 PyInline

PyInline [17] vznikl v roce 2001 z klávesnice Kena Simpsona. Tento modul pro Python umožňuje vkládat zdrojový kód C přímo do kódu Pythonu. Modul funguje tedy na podobném principu jako modul pro Perl, který napsal Brian Ingerson. Projekt je již bohužel několik let bez vývoje, avšak jsou i aktuální projekty, které ho používají. Dokumentace je strohá a mnoho funkcí není podporováno. PyInline nepodporuje programovací jazyk C++, ale pouze čistý programovací jazyk C. Zajímavostí však je, že takto napsaný kód se nemusí kompilovat, ale pouze se interpretuje. Na velké projekty se tento modul nehodí a spíše slouží k pochopení principu práce C API než k plnohodnotnému použití.

Jednoduchý příklad lze nalézt v příloze na straně 36.

2.3.4 Pyrex

Pyrex [23] vznikl v roce 2002 a jeho vývojářem je Greg Ewing. Knihovna je nástupcem výše uvedeného PyInline. Pyrex umožňuje psát kód pro převod mezi libovolnými datovými strukturami Pythonu a C. Pyrex využívá velmi zajímavou syntaxi kódu. Jelikož přebírá z jazyka Python jeho strukturu a metodiku psaní kódu (například žádné závorky) a využívá datové typy přejaté z jazyka C, jeví se jako dosti flexibilní nástroj. Právě díky těmto možnostem se dá snadno oblíbit. Škoda jen, že podporuje pouze programovací jazyk C. Pyrex má jako jeden z mála možnost generovat dokumentaci k modulu. Dokumentace lze generovat za pomocí nástroje pyrexdoc a dokumentace je vygenerována v podobě html dokumentu.

Jednoduchý příklad lze nalézt v příloze na straně 36.

2.3.5 Ctypes

Ctypes [28] je dalším modulem, který je zaštiťován samotnými vývojáři Pythonu. Jeho autorem je Thomas Heller a je licencován pod licencí MIT. Ctypes umožňuje volání funkcí z ddl knihoven/sdílených knihoven. Díky tomu je velmi jednoduché volání C callback funkcí v Pythonu. Na druhou stranu je tím pádem nemožné psát callback funkce přímo v Pythonu. Pro převod je ctypes velmi silný nástroj, bohužel pokud chce vývojář něco přidat, například další callback funkce napsané v Pythonu, tak je to zcela nemožné. Stejným problémem jsou i ukazatelé. Z těchto důvodů nelze použít ctypes pro napsání Verse modulu.

Jednoduchý příklad lze nalézt v příloze na straně 37.

2.3.6 Cython

Cython se vyvíjí od roku 2007 a jeho autory jsou Robert Bradshaw a Stefan Behnel. Cython převzal syntaxi z jazyka Python a je velmi podobný projektu Pyrex (vychází z něj). Oproti Pyrexu Cython umožňuje přímé volání C / C++ funkcí a definování C / C++ datových typů a je lépe optimalizován. Tyto dvě věci navíc umožňují generování C kódu přímo z Cythonu. Oproti ostatním projektům je Cython velmi rychlý a flexibilní. Další jeho

zajímavostí je možnost vygenerovat C funkce z funkcí napsaných v Pythonu. Díky těmto vlastnostem je Cython primárně využíván zejména k převodu zdrojových kódů Pythonu do C/C++. Cython využívají vývojáři například v Numpy [20] nebo Python Imaging Library (PIL) [25]. Pro vytvoření funkčního modulu je zapotřebí nejprve definovat funkce v Cython souboru a posléze spustit script *setup.py*, který je taktéž potřeba vytvořit.

Jednoduchý příklad lze nalézt v příloze na straně 37.

2.3.7 SWIG

SWIG [31] (Simplified Wrapper and Interface Generator) vznikl v roce 1996 jako open-source. Autorem a původním vývojářem byl Dave Beazley, který vyvinul SWIG, zatímco pracoval jako postgraduální student v Los Alamos. V současné době je vývoj SWIGu podporován aktivní skupinou dobrovolníků pod vedením Wiliama Fultona.

SWIG je po Ctypes a Boost:Python dalším z hodně známých a používaných programů pro psaní mezivrstev. Na rozdíl od výše uvedených využívá svůj vlastní pseudojazyk, který je oproti ostatním velmi dobře provedený. Mimo jiné SWIG podporuje pointery i C datové typy. Poradí si i s callback funkcemi, jak už implementovanými nebo nově vytvořenými v programovacím jazyce Python, které lze nalézt v dokumentaci. Jako jeden z mála, ne-li jediný, dokáže zaobalit C zdrojový kód i do jiných programovacích/scriptovacích jazyků. Vývoj na projektu je velice aktivní, občas by se dalo říci, že vývoj jde tak dopředu, že se zapomíná na dokumentaci. Starší dokumentace jsou o dost obsáhlejší a přesnější než nové. Z tohoto důvodu vzniká občas pochybnost, zda je nějaká funkce ve SWIGu implementována. Občas stačí jenom více pátrat a lze dané použití funkce najít v cizím příkladu. Z mnoha hledisek je nejlepší řešení zeptat se vývojářů na Internet Relay Chat (IRC).

Pro sestavení Python modulu je zapotřebí definovat funkce, které chceme převést v konfiguračním souboru SWIGu. Následně je nutné pomocí SWIGu vytvořit zaobalený kód v C API a Python soubor, který obsahuje funkce předělané pro Python. Posléze pomocí *GNU Compiler Collection (gcc)* [16] zkompilovat a vytvořit modul pro Python.

Jednoduchý příklad lze nalézt v příloze na straně 38.

3 Praktická část

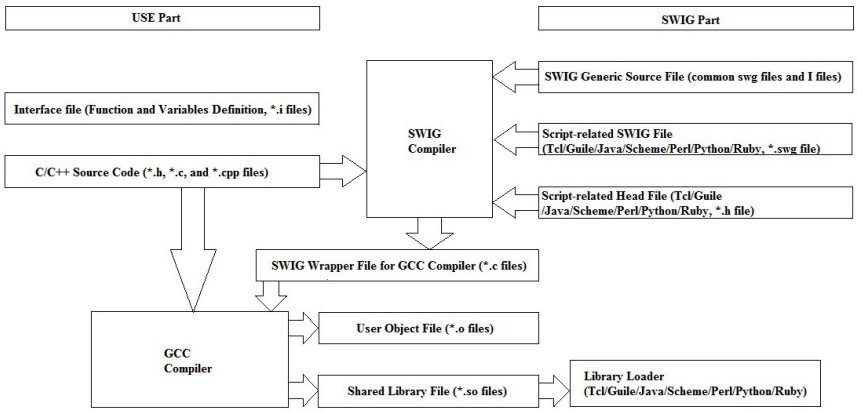
Pro vytvoření jednoduše použitelného modulu bylo potřeba použít techniku, která je flexibilní a co nejméně náročná na úpravy změn. Pro tyto požadavky vyhovuje nejlépe technika použití mezivrstvy. Při výběru knihovny pro mezivrstu bylo pohlíženo na několik kriterií. Bylo pohlíženo zejména na dokumentaci, rychlost tvorby modulu, pracnost změn v modulu a rychlost samotného modulu. Dalším zajímavým kritériem byla možnost používat modul i v jiných programovacích jazycích. Z popsaných programů v minulé kapitole byl nakonec vybrán SWIG. SWIG sice nemá nejaktuálnější dokumentaci, ale jeho ostatní vlastnosti vyvažují tento neduh.

3.1 SWIG

3.1.1 Seznámení

Jak bylo napsáno výše, SWIG slouží jako mezivrstva k převodu zdrojového kódu napsaného v C / C++ do jiného programovacího jazyka. Jako jeden z mála programů je SWIG možno použít jak na unixových operačních systémech, tak na operačních systémech Microsoft Windows (od Microsoft Windows 95). Princip fungování SWIGu lze demonstrovat na obrázku 1.

SWIG kompiluje mezivrstvy. Pro kompilaci potřebuje swigovské knihovny (*.swg) a hlavičkové soubory (*.h), které jsou dodávány se zdrojovými kódy SWIGu. Tyto soubory se mění v závislosti na vybraném jazyce, pro který kompilujeme mezivrstvu. Dalším nutným prvkem pro kompilaci mezivrstvy je tzv. ,,uživatelská část". Ta je definovaná zdrojovými kódy C / C++ (*.h,*.c a *.cpp) a tzv. ,,interface“ souborem (jeho funkce je vysvětlena v další kapitole). Takto vznikne tzv. ,,wrap" soubor, který je použit k vytvoření knihovny.



Obrázek 1: SWIG - princip činnosti při vytváření modulu pro Python

3.1.2 Interface soubor

Hlavním stavebním prvkem celého převodu je interface soubor. Interface soubor je rozdělen na tři hlavní části. Druhá a třetí část se mohou prolínat, však musí být odděleny složenými závorkami a každá část musí mít svou specifickou syntaxi (například pro vkládání souborů část pro kompilaci používá direktivu *#include* a část pro swig pro změnu používá direktivu *%include*).

V první části se uvádí název knihovny a vkládají se potřebné systémové knihovny (například podpora dokumentace nebo nedefaultních datových typů). Bez této části nelze vytvořit knihovnu a kompilace končí chybou.

Druhá část obsahuje informace pro kompilátor jazyka C / C++. Obsahuje hlavičkové soubory zdrojových kódů a funkce, pro které chceme vytvořit mezivrstvu. V této sekci lze taktéž deklarovat nové funkce a datové typy v jazyce C / C++, které budou obsaženy v nově vzniklé knihovně. Pakliže do druhé sekce nenapíšeme nic, kompilace se provede úspěšně, ale výsledná knihovna nebude nic umět. Tudíž se nedoporučuje nechávat ji prázdnou.

Ve třetí části se deklarují funkce, které budou v modulu dostupné. Tedy pokud se v druhé části uvedly veškeré funkce a ve třetí jenom nějaká část, pak programátor bude mít k dispozici pouze onu uvedenou část funkcí. Funkce se mohou lišit v konstruktoru

oproti druhé části, jenom musí být datové typy navzájem kompatibilní.

Taktéž se zde mohou uvádět nové funkce. Rozdíl oproti druhé části je však ten, že už je nutné psát za pomoci syntaxe wrapu. Syntaxe wrapu je ovlivněna knihovnou jazyka, pro který chceme vytvořit knihovnu (v případě této bakalářské práce tedy Python objectů). Toto specifikum tudíž dost často dělá z interface souborů velmi nepřehledné soubory, které obsahují mnohdy i několik stovek řádků kódu. V této sekci se mohou stejným způsobem definovat i datové struktury, datové typy, callback funkce a jejich filtry, jejich deklarace je ale mnohem složitější a jejich vytváření bude vysvětleno a demonstrováno v dalších kapitolách (zejména callback funkce).

3.1.3 Kompilace

Abychom mohli vytvořit použitelnou knihovnu nebo modul, je potřeba provést kompilaci, která probíhá ve třech krocích. V prvním kroku se vytvoří ,,wrap" - soubor za pomocí SWIGu, který obsahuje převedené funkce z C / C++ (uvedené v interface souboru) do C API. Princip provedení byl popsán v kapitole pojednávající o SWIGu. V druhém kroku použijeme kompilátor pro překompilování původního zdrojového kódu a wrapu. Třetí krok slouží k vytvoření dynamické knihovny, která tvoří vlastní modul. Pro kompilaci byl použit standartní komplilátor *gcc*, ale lze použít i jiný standartizovaný kompilátor jazyka C / C++. Kvůli zrychlení kompilovaní byl vytvořen Makefile [19], který obsahuje veškeré informace pro kompilátor. Díky tomu můžeme zkompilovat modul za pomoci jednoho příkazu a pokud bychom chtěli kompilovat na jiném stroji, stačí pozměnit cesty k potřebným knihovnám. Pro vytvoření dynamické knihovny (nutnost pro vytvoření modulu) byl použit program ld [38], který je součástí každé linuxové distribuce.

Jednoduchý příklad:

```
$ swig -python example.i
$ gcc -c example.c example_wrap.c \
      -I/usr/local/include/python2.7
$ ld -shared example.o example_wrap.o -o _example.so
```

V příkladu je jednoduchá demonstrace provedení kompilace a vzniku modulu pro programovací jazyk Python. První řádek spouští SWIG, který vytvoří potřebnou mezivrstvu (wrap). Je nutno si povšimnout, že je SWIG spouštěn s parametrem ,,-python“ tento parametr lze vyměnit za jiný a tím lze změnit cílový programovací jazyk. Druhý parametr uvadí název interface souboru, který obsahuje všechna potřebná data pro vznik wrapu. Druhý příkaz je spuštění kompilátoru gcc. Byly použity jenom nutné parametry. Tedy původní zdrojový kód C / C++ a námi vytvořená mezivrstva a jako poslední parametr bylo nutné uvést cestu ke knihovnám daného cílového programovacího jazyka, v našem případě Python. Poslední příkaz vytváří z námi vytvořené knihovny dynamickou knihovnu. Výstupem bude už výsledná dynamická knihovna, která bude použita pro psaní kódu již v Pythonu.

3.1.4 Datové typy a konstanty

SWIG umožňuje používat základní datové typy [32], které vycházejí z programovacího jazyka C: int, short, long, unsigned short, unsigned long, unsigned char, signed char, boolean, float, double.

Díky této skutečnosti lze tyto datové typy aplikovat a používat i v ostatních jazycích po převodu (například: unsingned long zůstane i v perlu jako unsingned long a nestane se z něj jenom long). Pokud by byla potřeba využívat i jiné datové typy, než jsou v základu k dispozici (například uint32), tak je možnost jejich podporu do SWIGu zakomponovat pomocí přídavných knihoven. Knihovny lze stáhnout na stránkach projektu SWIGu, případně více specializované knihovny lze nalézt v mailing listu. Škoda jen, že většina knihoven je určena pro starší verze SWIGu a tedy v novějších verzích jsou nefunkční nebo fungují nekorektně, což komplikuje případnou práci s těmito knihovnami. Bohužel takto přidané datové typy lze využívat jen při práci s funkcemi, které se převádí. Nelze je používat při testování vstupních dat do funkcí, což je mnohdy nežádoucí. Pokud je nutné testovat vstupní data a je použit datový typ, který nelze použít, pak je nutné přepsat (zobecnit) datový typ v daných C zdrojových kódech (například: datový typ uint32 se musí zobecnit na int).

Pakliže je nutné použít datový typ, který není podporován programovacím jazykem, pro který je knihovna určena, je nutné použít pointery. Pomocí nich lze nadefinovat v interface souboru SWIGu nové funkce, které pracují s pointery a umožnují převod podporovaných datových typů na nepodporované, ale potřebné datové typy. Je však nutné při programování myslet na to, že při práci s hodnotami těchto typů je nutno přistupovat k těmto hodnotám jenom pomocí funkcí k tomu určených.

Konstanty lze definovat pomocí příkazu *#define*. Jinak je definice a chování stejné jako v jiných programovacích jazycích. Tedy jejich název je definován velkými písmeny a jejich hodnoty nelze nikterak změnit.

Jednoduchý příklad použití konstant a proměnných:

soubor example.i

```
%module example
double a;                   // definice promenne
#define PI      3.14159     // definice PI konstanty
```

Následně se provede kompilace knihovny (viz předešlá kapitola) a po spuštění Pythonu lze knihovnu vyzkoušet takto:

```
>>> import example
>>> example.a = 4.3       // nastaveni hodnoty
>>> print example.a       // vypsani hodnoty
>>> 4.3
>>> print example.PI      // vypsani konstanty PI
>>> 3.14159
```

3.1.5 Záznamy

Záznamy jsou složené datové typy definované programátorem. SWIG umožňuje používat již vytvořené záznamy (v hlavičkových souborech jazyka C / C++) a také nabízí možnost vytvoření nových datových struktur. V níže uvedeném příkladu je ukázka definice jednoduchého záznamu - vektoru. Na něm je vidět, jak je samotný záznam definován a jak

je nutné používat SWIG syntaxi. Pro každou akci s vektorem je taktéž nutné napsat k nim patřičné funkce. V příkladu jsou ukázany funkce pro nastavení a vypsání hodnoty dané souřadnice vektoru a vymazání vektoru.

Jednoduchý příklad (převzat z dokumentace SWIGu [33]):

Nejprve je nutné definovat záznam v interface souboru.

soubor example.i

```
typedef struct {
        double x,y,z;
} Vector; //definice vektoru
/* get a set funkce */
double Vector_x_get(struct Vector *obj) { return obj->x; }
double Vector_y_get(struct Vector *obj) { return obj->y; }
double Vector_z_get(struct Vector *obj) { return obj->z; }
void Vector_x_set(struct Vector *obj, double value)
{ obj->x = value; }
void Vector_y_set(struct Vector *obj, double value)
{ obj->y = value; }
void Vector_z_set(struct Vector *obj, double value)
{ obj->z = value; }
/* konstruktor */
struct Vector *new_Vector() {
    return (Vector *) calloc(1,sizeof(struct Vector));
}
/* destruktor */
void delete_Vector(struct Vector *obj) { free(obj); }
```

Z výše uvedených řádků je vidět, že definice konstruktorů a destruktorů vychází z programovacího jazyka C. Po kompilaci knihovny můžeme s knihovnou pracovat v Pythonu následovně:

```
>>> import example
>>> v = example.new_Vector()
>>> example.Vector_x_set(v,2)
>>> example.Vector_y_set(v,10)
>>> example.Vector_z_set(v,-5)
...
>>> example.delete_Vector(v)
```

3.1.6 Ukazatele

Ukazatele [4], neboli pointery, jsou nedílnou součástí programovacího jazyka C. Proto bylo nutné při tvorbě modulu implementovat i je. Po instalaci SWIGu je k dispozici pouze podpora ukazatele, který se převádí z jazyka C. Nejdůležitější informací je, že se nemusí takto předaný ukazatel jakkoliv alokovat, takže se převod takových funkcí stává maličkostí.

Jednoduchý příklad:

```
extern void foo(int *ptr);
```

Pokud ovšem chceme používat ukazatele ve vlastních funkcích definovaných ve SWIGu nebo ve filtrech pro callback funkce, tak musíme využít modul, který je dodáván ve standartním instalačním balíčku. Bohužel modul je použitelný pouze pro dva základní datové typy. A to pro datový typ int a double. Jenom tyto dva datové typy však na všechno nestačí. V dokumentaci je psáno, že existuje rozšiřující modul, který rozšiřuje základní modul o další datové typy (například unsigned int32, char). Po snaze tento modul zprovoznit, lze říci, že modul v nové verzi SWIGu nefunguje a jeho vývoj již přibližně rok nepokračuje. Díky této skutečnosti bylo buď nutné upravit několik funkcí v původních zdrojových kódech (*vc_connection.c* a *verse.h*) a nebo si tyto funkce, které pracujicí s ukazateli napsat ve SWIGu.

V první verzi knihovny byla použita první možnost. S následnou změnou struktury některých proměnných a funkcí v původních zdrojových kódech protokolu Verse bylo nutné datové struktury v interface souboru pozměnit. To se ukázalo jako velmi zdlouhavá a chy-

bová činnost. Proto byla zvolena pro novou knihovnu druhá možnost. Nová verze Verse používá místo uint32 a int v dosti případech uint8, což lze snadno nadefinovat za pomoci unsigned charu. Díky této skutečnosti bylo vytvoření potřebných funkcí pro práci s ukazatelem typu uint8 celkem jednoduché.

V níže uvedeném příkladu je demonstrována ukázka deklarace použití ukazatele, který se hojně používá při programování, a následná práce s takto vytvořeným ukazatelem (vytvoření a získání hodnoty).

Jednoduchý příklad:

soubor example.c

```
void add(int a, int b, int *c)
     { *c = a + b; }
```

soubor example.i

```
%module example
%include "cpointer.i"          //vlozeni modulu cpointer
%pointer_functions(int, intp); //inicializace ukazatele

void add(int a, int b, int *c);//deklarace funkce
                               //z example.c
```

Po zkompilování již lze pracovat s modulem následovně:

```
>>> import example
>>> c = example.new_intp()     # inicialization pointer
>>> example.add(1,2,c)         # function call
>>> example.intp_value(c)      # print c value
3
>>> example.delete_intp(c)     # call pointer destructor
```

3.1.7 Callback funkce

Pro vytvoření Verse klienta je nezbytně nutné mít možnost definovat v programovacím jazyce Python vlastní callback funkce. SWIG sice umožňuje použití callback funkcí a v dokumentaci je i několik názorných ukázek, ale jak jde vývoj SWIGu kupředu, tak jsou tyto informace zastaralé a mnohdy i nefunkční. Ve SWIGu se callback funkce deklarují pomocí ,,pseudo funkcí“, které jsou obecné prototypy těchto funkcí.

Jednoduchá ukázka:

soubor example.c

```
int do_func(int a, int b, int (*func)(int,int)) {
  return (*func)(a,b);
}
int add(int a, int b) {
  return a+b;
}
```

soubor example.i

```
%module example
%{
#include "example.h"
%}
extern int  do_func(int a, int b, int (*func)(int, int));
%constant int (*ADD)(int,int) = add;//registrace preudo funkce
extern int (*funcvar)(int,int);     //deklarace pseudo funkce
```

soubor example.h

```
extern int do_func(int,int, int (*func)(int,int));
extern int add(int,int);
```

```
>>> import example
>>> a = 6
>>> b = 2
/* zavolani callback funkce */
>>> print "ADD(a,b) =", example.do_func(a,b,example.ADD)
>>> ADD(a,b) = 8
```

Výše uvedený příklad demonstruje pouze použití callback funkcí, které již byly napsány v jazyku C. Jak lze z příkladu vidět, tak takto deklarované callback funkce je jednoduché přidat do nové knihovny. Avšak takto vytvořené callback funkce postrádají skoro celý svůj smysl. Při tvorbě modulu pro Verse je nutné mít možnost zaregistrovat vlastní callback funkce napsané v Pythonu. A to z toho důvodu, aby měl vývojář možnost ovlivňovat svůj program pomocí callback funkcí. Tato možnost byla stěžejní při tvorbě Verse modulu pro Python. Nejprve je nutné v interface souboru definovat funkce, které registrují callback funkce napsané v jazyce C a k nim přiřadit funkce, které registrují callback funkce napsané v Pythonu. Tyto funkce musí být psány SWIG syntaxí. Pro správnou funkčnost těchto funkcí je nutné napřed volanou callback funkci zkontrolovat filtrem. Filtr zaručuje to, že se nepošlou data se špatnou datovou strukturou. Tedy se musí testovat, zda danou callback funkci lze volat a zároveň, zda datové typy odpovídají datovým typům. Pro příklad je níže uveden výňatek z interface souboru pro Verse modul.

soubor example.i

```
//deklarace PyObjektu pro callback funkci
static PyObject *my_pycallback = NULL;
//funkce pro prevod parametru z callback funkce
static void py_register_receive_user_authenticate_CallBack(
const uint8 session_id,const char *username,
const uint8 auth_methods_count, const uint8 *methods)
{
   PyObject *func, *arglist;
   PyObject *result;
   func = my_pycallback;
   arglist = Py_BuildValue("csis",session_id,username,
                           auth_methods_count,methods);
   PyObject_Print(arglist, stdout, 0);
   result = PyEval_CallObject(func,arglist);
   Py_DECREF(arglist);
   Py_XDECREF(result);
   return;
}
static void py_register_receive_user_authenticate(PyObject *PyFunc)
{ Py_XDECREF(cb3);
  Py_XINCREF(PyFunc);
  cb3 = PyFunc;
  register_receive_user_authenticate(
   py_register_receive_user_authenticate_CallBack);
}
//definovani viditelne funkce pro zaregistrovani  CB
void py_register_receive_user_authenticate(PyObject *PyFunc);
```

V příkladu je nutné si všimnout několika zajímavých částí. První zajímavost

je *Py_BuildValue*. Tento příkaz zajišťuje kontrolu vstupních datových proměnných a zároveň provádí převod datových typů pro původní callback funkci. Další zajímavost je *Py_DECREF* a *Py_XDECREF*. Tyto dva příkazy se starají hlavně o práci s pamětí. Bez těchto příkazů by při registraci více callback funkcí docházelo k přepisování vrcholu zásobníku, tedy by callback funkce nefungovaly nebo by způsobovaly náhodné pády a nebo dokonce by pracovaly s daty, která jim nenáleží. Okolnosti ohledně výběru z těchto chyb záleží z podstatné části na programovacím jazyku, pro který je callback funkce určena. Vhodné je taktéž použít pro každou callback funkci vlastní objekt, sice by se dalo používat pouze jeden objekt a ten pomocí výše uvedených funkcí přepínat, ale v praxi to nefunguje na 100% tak, jak by teoreticky mělo.

3.2 Textový klient

Textový klient byl napsán k vyzkoušení, zda modul vytvořený ve SWIGu pracuje tak, jak se od něj očekává. Celý kód se nachází na CD, které je přiloženo.

Při psaní klienta byly napsány jednoduché callback funkce, aby díky nim šel následně lépe diagnostikovat modul. Po první verzi a testu callback funkcí bylo zjištěno, že callback funkce jsou sice zaregistrované, ale že se jim nepředávají správně argumenty funkcí. Aby tato chyba byla napravena, musela se dodělat do SWIGu kontrola prototypu callback funkcí a testování vstupních argumentů [36] (jelikož tato možnost nebyla ošetřena v původním kódu).

Pro samotné spuštění textového klienta stačí zadat příkaz *python client-tui.py*. Pokud dostane klient odpověď od serveru, pak stačí zadat jenom uživatelské jméno a posléze i heslo. Po přihlášení probíhá řetězové spouštění všech implementovaných funkcí v modulu. Tím se testuje, zda všechny funkce (hlavně callback funkce) pracují tak, jak by měly. Po skončení všech funkcí se klient odhlásí a ukončí. Všechny akce klienta lze sledovat v konzoli, kde se vypisují „DEBUG" hlášky, které informují o každé kroku, který program vykonal.

Na obrázku 2 je vidět, jak se po přihlášení vytváří inicializační uzly (ukázka, že modul pracuje správně). Lze si taktéž povšimnout, že každý uzel si s sebou nese několik důležitých

informací [11].

```
DEBUG: Send packet: Socket: 4, [::1]:20000 Ver: 1, Flags: PAY,ACK,ANK, Window:0, PayID:803739200, AckNakID:2, AnkID:803739199
        ACK COMMAND, 1275537884
DEBUG: Adding packet: 803739200 to history
        NODE_CREATE, NodeID: -1, ParentID: 65537, UserID: 1000
        NODE_SUBSCRIBE, NodeID: 0, level: 1
        NODE_SUBSCRIBE, NodeID: 1, level: 0
DEBUG: send() 53 bytes
DEBUG: Receive packet: Socket: 4, bufsize: 185, [::1]:20000 Ver: 1, Flags: PAY,ACK,ANK, Window:0, PayID:1275537885, AckNakID:2, AnkID:1275537884
        ACK COMMAND, 803739200
        NODE_CREATE, NodeID: 65538, ParentID: 65537, UserID: 1000
        NODE_CREATE, NodeID: 1, ParentID: 0, UserID: 100
        NODE_CREATE, NodeID: 65537, ParentID: 1, UserID: 100
        NODE_CREATE, NodeID: 2, ParentID: 0, UserID: 100
        NODE_CREATE, NodeID: 100, ParentID: 2, UserID: 100
        NODE_CREATE, NodeID: 1000, ParentID: 2, UserID: 100
        NODE_CREATE, NodeID: 3, ParentID: 0, UserID: 100
        NODE_TEST, NodeID: 0, Frame: 0, Pos:  0.000,  0.000,  0.000
        NODE_TEST, NodeID: 1, Frame: 0, Pos:  0.000,  0.000,  0.000
        NODE_TEST, NodeID: 2, Frame: 0, Pos:  0.000,  0.000,  0.000
        NODE_TEST, NodeID: 3, Frame: 0, Pos:  0.000,  0.000,  0.000
DEBUG: Removing packet: 803739200 from history
```

Obrázek 2: Verse textový klient napsaný v Pythonu - ukázka registrace nových uzlů

Textový klient byl stavebním kamenem grafického klienta, který bude popsán v následující kapitole. Slouží hlavně jako testovací prostředek a k pochopení fungování síťového protokolu Verse.

3.3 Grafický klient

3.3.1 Grafické rozhraní

Grafický klient, na rozdíl od textového, neměl jen prezentovat funkčnost modulu, ale také měl poukázat na přednosti Verse protokolu. Grafického Klienta lze použít jako obecného správce dat na serveru, což bylo použito při testovaní jak modulu Verse klienta pro Python, tak i Verse serveru. Při psaní grafického uživatelského rozhraní (GUI) je nutno použít modul, který je k tomu určen. Mezi nejznámější patří Tkinter [37], PyQT a PyGTK.

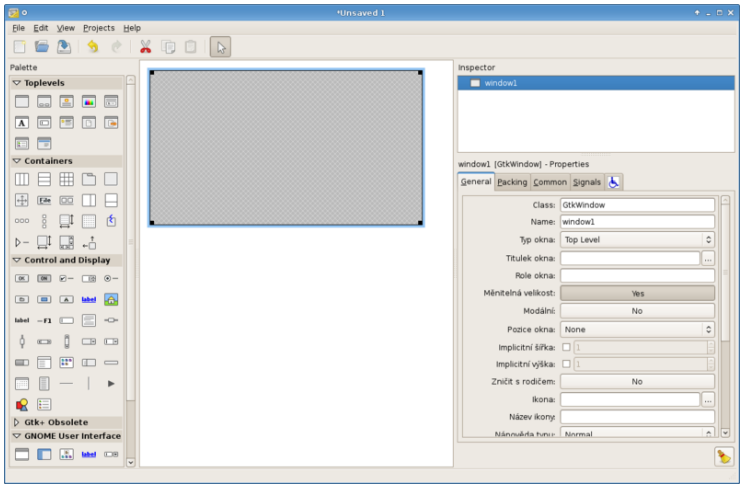
Tkinter je modul, který je již obsažen v každé instalaci Pythonu. Tkinter umožňuje programátorovi vytvářet klasické okenní aplikace, jak je na ně zvyklý z Windows, GNOME či KDE. Bohužel Tkinter neobsahuje veškeré vymoženosti PyQT a PyGTK, proto se hodí na jednodušší projekty.

PyQT a PyGTK jsou moduly, které vycházejí ze svých knihoven, tedy pro PyGTK to je knihovna GTK+ (GIMP Toolkit) a pro PyQT knihovna QT [27]. Knihovna QT i knihovna GTK+ umožňují vytvořit GUI dle představ programátora. Avšak QT je framework pro tvorbu desktopových a mobilních aplikací, kdežto GTK+ je pouze knihovna pro tvorbu

GUI. QT je používáno hlavně s desktopovým prostředím KDE (K Desktop Environment) a GTK+ je spojeno hlavně s desktopovým prostředím GNOME (GNU Network Object Model Environment). Obě knihovny jsou multiplatformní (i když PyGTK trošku zaostává před PyQT na platformě Windows). U každé z nich má programátor možnost psát kód pro GUI do samotného programu nebo má možnost využít GUI designer. Z těchto modulů byl nakonec vybrán modul PyGTK pro jeho provázánost s GNOME a s autorovými předešlými zkušenostmi s GTK+ knihovnami při psaní aplikací.

Při psaní grafického klienta bylo nutné rozhodnout, zda bude kód pro GUI obsažen v samotném kódu aplikace nebo odděleně. V první verzi grafického klienta byla použita první varianta. Při ní bylo zjištěno, že kód se stává méně čitelným a hůře se v něm orientovalo, zejména při změnách komponent v GUI aplikace. Proto byla pro druhou verzi grafického klienta použita druhá varianta. Příklady variant lze nalézt v přílohách na straně 32.

Jako grafické rozhraní pro tvorbu GUI byl použit GLADE [15]. Glade je svobodný software uvolnění pod GNU General Public License, který se vyvíjí od roku 1998. K tvorbě gui používá GladeXML, což je formát XML [9]. Tato vlastnost umožňuje dynamicky načítat GUI při běhu aplikace, což usnadňuje překládání do jiných jazyků.

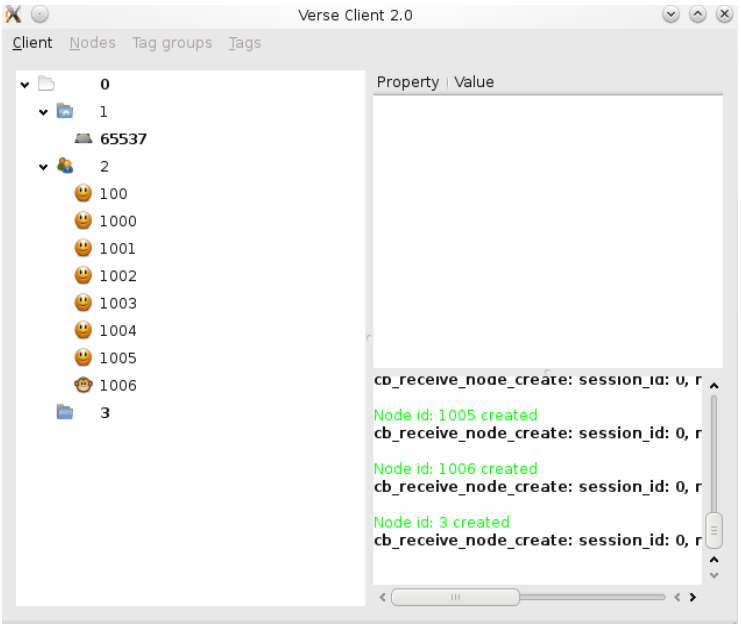


Obrázek 3: GLADE - ukázka pracovního prostředí

Z obrázku 3 je patrné, že prostředí GLADE má klasické rozložení komponent. Tedy nalevo lze nalézt paletu komponent, které se dají pomocí přetáhnutí přenést na pracovní plochu, kde se komponenta usadí na daný formulář. Napravo se nachází panel, který zobrazuje stromovou sturkturu GUI. Pod ním lze nastavit vlastnosti aktivní komponenty.

3.3.2 Grafický klient

Po spuštění klienta je nejprve nutné přihlásit se k Verse serveru. Po přihlášení k serveru se aktivují nabídky k tvorbě uzlů, tagů a skupiny tagů. Na obrázku 4 lze vidět, jak vypadá aplikace po přihlášení k Verse serveru. Na levé straně je stromová struktura, která obsahuje

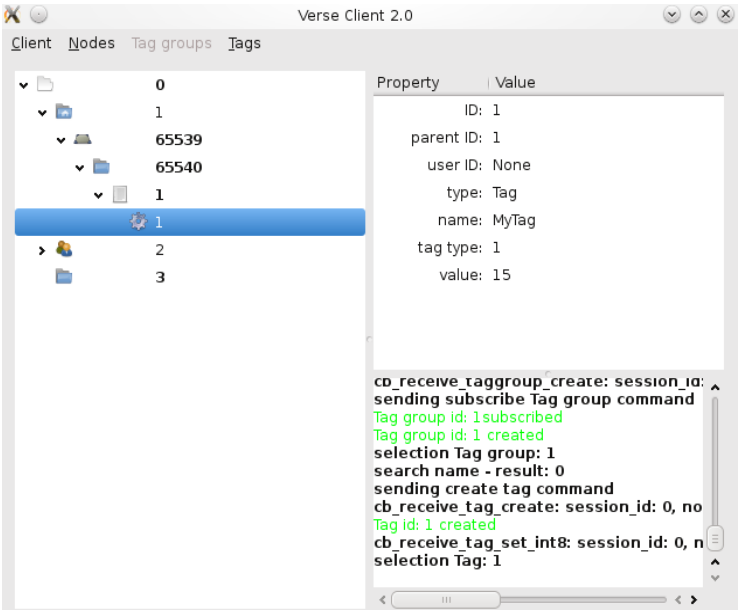


Obrázek 4: Verse klient - ukázka pracovního prostředí

veškeré uzly, skupiny tagů a tagy, které nám buď poslal server nebo které jsme vytvořili - viz obrázek 4 . Každá ikona reprezentuje určitý uzel. Uzly, které jsou tučně zvýrazněné, jsou uzly, ke kterým je daný uživatel přihlášen (příkazy *verse_send_node_subscribe()* a *verse_send_taggroup_subscribe()*). Prvky, ke kterým není uživatel přihlášen, jsou označeny netučným písmem. Přihlásit a odhlásit se lze pouze od uzlu a od skupiny tagů.

K práci s daty slouží dvě nabídky. První je klasické menu, které se nachází v horní části aplikace, druhé je tzv. ,,vyskakovací menu“, které lze vyvolat zmáčknutím pravého tlačítka na myši. Aktivní je vždy menu dle daného typu prvku, který máme vybraný ve stromové struktuře.

Informace o daném prvku lze nalézt po nakliknutí v pravé horní části, kde se nacházejí veškeré dostupné informace - ukázka na obrázku 5.

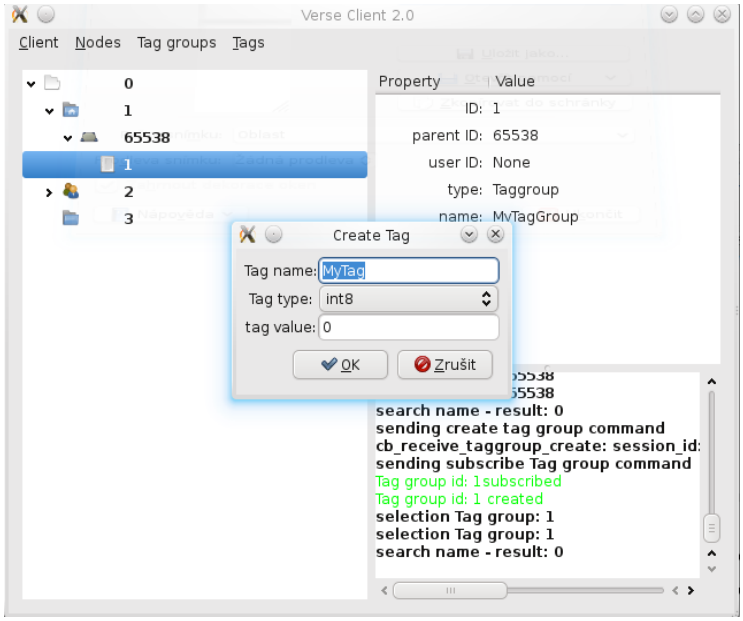


Obrázek 5: Verse klient - informace vybraného tagu

V dolní části si lze povšimnout informativního boxu. V něm se nacházejí informace o provedených akcích, případně se zde vypisují chyby aplikace nebo příkazu. Informace jsou tří typů - ,,debug” (tučné písmo), ,,success“ (zelené písmo), ,,error” (červené písmo). Debug výpisy je možno vypnout v nastavení aplikace.

Grafický klient umožňuje vytváření uzlů a skupin tagů a tagů. Uzel se vytvoří vždy do uzlu avatara přihlášeného uživatele. Při tvorbě skupiny tagů je nutné mít označený uzel, který bude rodičem dané skupiny tagů. Tato podmínka platí i při tvoření tagu, ale musí být označena skupina tagů. Při těchto akcích se vždy zobrazí nový dialog, ve kterém je nutné zadat jméno pro daný prvek. Aplikace sama přidává číslování k prvkům, které by

měly mít stejné jméno. Při tvorbě tagu je nutné zvolit ještě typ tagu a jeho hodnotu - ukázka na obrázku 6. Uživatel může taktéž dané prvky smazat, avšak může smazat jedině



Obrázek 6: Verse klient - ukázka tvorby tagu

prvky, které sám vytvořil nebo je jejich vlastníkem. Pakliže uživatel smaže prvek, který je rodičem dalších prvků, pak tyto prvky budou taky odstraněny. Protokol Verse má však prozatím neduh, že nelze vytvořit nový tag nebo skupinu tagů se jménem, které již bylo použito, ačkoliv daný prvek byl již smazán.

Tag je jediný prvek, u kterého lze měnit vlastnosti po vytvoření. Lze u něj změnit hodnotu. Pro změnu hodnoty se zobrazí dialog, ve kterém je možno real-time měnit hodnotu tagu. Změna se projevuje automaticky u všech uživatelů, kteří jsou přihlášeni k danému prvku.

Modul Verse umožňuje pomocí příkazu *verse_node_link()* přemístit uzel, který uživatel vlastní do uzlu jednotlivých 3D scén (uzel s ID = 3). Tato operace se v grafickém klientu dá provést pomocí operace drag & drop. Operace se provede pokud se splní podmínky. První podmínkou je, že vybraný uzel k přesunu je vlastněn uživatelem, druhá podmínka je, že uzel, který se má stát rodičem je buďto uzel jednotlivých 3D scén nebo avatar uživatele.

4 Závěr

Pro implementaci modulu klientské části protokolu Verse byla vybrána metoda tvorby modulu za pomocí tzv. mezivrstvy. Mezivrstvou byl zvolen program SWIG. SWIG byl vybrán pro jeho rozsáhlou dokumentaci a možnosti, díky kterým lze vytvořit rychlý a snadno upravitelný modul. Modul je plně funkční a obsahuje veškeré potřebné funkce k napsání plnohodnotného klienta v Pythonu. Jelikož SWIG ještě úplně nepodporuje Python 3.1 a vyšší, tak byl modul napsán pro Python 2.7. Díky flexibilitě skriptů pro kompilaci je možné už teď vytvořit moduly nebo knihovny i pro další programovací jazyky. Jakmile SWIG doplní úplnou podporu pro Python 3.1, pak bude možné používat modul beze změn i pro Python v této verzi. K demonstraci funkčnosti Python modulu byl naprogramován textový a grafický klient. Textový klient testuje veškeré funkce knihovny. Grafický klient předvádí možnosti Verse protokolu. Pro grafického klienta byla použita grafická knihovna GTK+. Při testování modulu se podařilo odhalit několik chyb v implementaci Verse serveru i samotné knihovny. Tyto chyby se už podařilo z větší části odstranit. Nedostatkem modulu je nemožnost kontroly vstupních datových typů funkcí, jelikož to pro některé datové typy SWIG nepodporuje. Na tomto nedostatku by mohlo být zapracováno a mohly by být implementovány další funkce protokolu Verse. Modul obsahuje vlastní nápovědu k jednotlivým funkcím a v příloze je k dispozici dokumentace.

K bakalářské práci je přiloženo CD, na kterém se nachází jak veškeré zdrojové kódy potřebné ke kompilaci Python modulu protokolu Verse, tak hotové aplikace, které byly uváděné v bakalářské práci.

Reference

[1] *PILGRIM, Mark. Dive into Python. New York: Distributed to the Book trade in the United States by Springer-Verlag, c2004, 413 s. ISBN 15-905-9356-1.*

[2] *SUMMERFIELD, Mark. Python 3: výukový kurz. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 584 s. ISBN 978-80-251-2737-7.*

[3] *MASTERS, Jon a Richard BLUM. Linux profesionálně: programování aplikací. Vyd. 1. Brno: Zoner Press, 2008, 539 s. ISBN 978-80-86815-71-8.*

[4] *HEROUT, Pavel. Učebnice jazyka C. 4., přeprac. vyd. České Budějovice: Kopp, 2004, s. 144-169. ISBN 80-7232-220-6.*

[5] *HNÍDEK, Jiří. Síťový protokol pro grafické aplikace. Liberec, 2010. Disertační práce. Technická univerzita v Liberci.*

[6] *DIERKS, T., and ALLEN, C. The TLS Protocol Version 1.0. RFC 2246, IETF, jan 1999. http://www.ietf.org/rfc/rfc2246.txt, Obsoleted by RFC 4346, updated by RFCs 3546, 5746.*

[7] *POSTEL, J. RFC 793: Transmission Control Protocol. RFC 793, IETF, sep 1981. http://www.ietf.org/rfc/rfc793.txt, Updated by RFCs 1122, 3168.*

[8] *POSTEL, J. RFC 768: User Datagram Protocol. RFC 768, IETF, aug 1980. http://www.ietf.org/rfc/rfc768.txt.*

[9] *Extensible Markup Language [online]. 2008 [cit. 2012-03-13]. Dostupné z WWW:* `<http://www.w3.org/TR/2008/REC-xml-20081126/REC-xml-20081126.xml>`.

[10] *HNÍDEK, J. New Verse API [online]. 5. 8. 2010 [cit. 2012-03-20]. Dostupné z WWW:* `<http://www.nti.tul.cz/en/WikiUser:Jiri.Hnidek/New_Verse_API.en>`.

[11] *HNÍDEK, J. Proposal of New Verse Protocol [online]. 2010 [cit. 2012-03-13]. Dostupné z WWW:* <http://www.nti.tul.cz/en/WikiUser:Jiri.Hnidek/Verse_resend_mechanism.en>.

[12] *Blender [online]. 20. 3. 2012 [cit. 2012-03-20]. Dostupné z:* <http://www.blender.org/>.

[13] *Boost::Python [online]. 26.3.2003 [cit. 2012-03-20]. Dostupné z WWW:* <http://www.boost.org/>.

[14] *Cython [online]. 20011 [cit. 2012-03-16]. Dostupné z WWW:* <http://cython.org/>.

[15] *Glade - A User Interface Designer [online]. 2009 [cit. 2012-03-21]. Dostupné z WWW:* <http://glade.gnome.org/>.

[16] *GNU Compiler Collection [online]. 2012 [cit. 2012-03-12]. Dostupné z WWW:* <http://gcc.gnu.org/>.

[17] *Inline for Python [online]. 2001 [cit. 2012-03-20]. Dostupné z WWW:* http://pyinline.sourceforge.net/

[18] *IronPython [online]. 2012 [cit. 2012-03-15]. Dostupné z WWW:* <http://www.ironpython.net/>.

[19] *Linuxsoft.cz [online]. 21.4.2005 [cit. 2012-03-03]. C/C++ (17) - Makefile. Dostupné z WWW:* <http://www.linuxsoft.cz/article.php?id_article=722>.

[20] *Numpy [online]. 2009 [cit. 2011-05-12]. Dostupné z WWW:* <http://numpy.scipy.org/>.

[21] *PyGTK: GTK+ for Python [online]. 1. 4. 2011 [cit. 2012-03-20]. Dostupné z WWW:* <http://www.pygtk.org/>.

[22] *PyQt [online]. 2010 [cit. 2012-03-20]. Dostupné z WWW:* <http://www.riverbankcomputing.co.uk/software/pyqt>

[23] *Pyrex - a Language for Writing Python Extension Modules [online]. 12.4.2010 [cit. 2012-03-21]. Dostupné z WWW:* <http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/>.

[24] *Python/C API Reference Manual [online]. 2012 [cit. 2012-03-30]. Dostupné z WWW:* <http://docs.python.org/c-api/>.

[25] *Python Imaging Library [online]. 2009 [cit. 2012-03-12]. Dostupné z WWW:* <http://www.pythonware.com/products/pil/>.

[26] *Python Programming Language [online]. 2012 [cit. 2012-03-15]. Dostupné z WWW:* <http://http://www.python.org/>.

[27] *QT [online]. 2012 [cit. 2012-03-20]. Dostupné z WWW:* <http://qt.nokia.com/products/>

[28] *Scipy.org [online]. 2009 [cit. 2012-03-15]. Cookbook / Ctypes. Dostupné z WWW:* http://www.scipy.org/Cookbook/Ctypes

[29] *SIP Reference Guide [online]. 1.2.2011 [cit. 2012-03-20]. SIP Reference Guide. Dostupné z WWW:* <http://www.riverbankcomputing.com/static/Docs/sip4/index.html>

[30] *STEENBERG, E. BRINK, E. The Verse Specification [online]. 2007 [cit. 2012-03-21]. Dostupné z WWW:* <http://verse.blender.org/>.

[31] *SWIG [online]. 2.6.2010 [cit. 2012-03-10]. Dostupné z WWW:* <http://www.swig.org>.

[32] *SWIG2.0 Documentation [online]. 1.1.2010 [cit. 2012-03-03]. Basic Type Dostupné z WWW:* http://www.swig.org/Doc2.0/SWIGDocumentation.html#SWIG_nn10

[33] *SWIG2.0 Documentation [online]. 1.1.2010 [cit. 2012-03-03]. Structures and unions Dostupné z WWW:* <http://www.swig.org/Doc2.0/SWIGDocumentation.html#SWIG_nn31>

[34] *The GTK+ project [online]. 2008 [cit. 2012-03-13]. Dostupné z WWW:* <http://www.gtk.org/>.

[35] *The Jython Project [online]. 2012 [cit. 2012-03-15]. Dostupné z WWW:* <http://www.jython.org/>.

[36] *The Py_BuildValue() Function [online]. 2010 [cit. 2012-03-03]. Dostupné z WWW:* <http://docs.python.org/release/2.0.1/ext/buildValue.html>.

[37] *Tkinter [online]. 2012 [cit. 2012-03-15]. Dostupné z WWW:* <http://docs.python.org/library/tkinter.html>.

[38] *Unix Command: ld [online]. 2011 [cit. 2012-03-13]. Dostupné z WWW:* <http://linux.about.com/library/cmd/blcmdl1_ld.htm>.

5 Ukázky tvorby GUI

5.1 Ukázka GUI psané v kódu aplikace

```
import pygtk
pygtk.require("2.0")
import gtk
# create new window
window = gtk.Window()
# connect event for close application
window.connect("delete-event", gtk.main_quit)
# label with text hello world
label = gtk.Label("Hello world")
# connect to main window
window.add(label)
window.show_all()
gtk.main()
```

5.2 Ukázka GUI s pomocí GLADE

Předpokladem této ukázky je to, že máme za pomoci GLADE vytvořené již GUI, které je uloženo v souboru helloworld.glade.

```
#!/usr/bin/env python


import sys
try:
         import pygtk
          pygtk.require("2.0")
except:
          pass
try:
```

```
        import gtk
          import gtk.glade
except:
        sys.exit(1)

class HellowWorldGTK:
        """This is an Hello World GTK application"""

        def __init__(self):

                #Set the Glade file
                self.gladefile = "helloworld.glade"
                self.wTree = gtk.glade.XML(self.gladefile)

                #Create our dictionay and connect it
                dic = { "on_btnHelloWorld_clicked" : self.btnHelloWorld_clicked,
                        "on_MainWindow_destroy" : gtk.main_quit }
                self.wTree.signal_autoconnect(dic)

        def btnHelloWorld_clicked(self, widget):
                print "Hello World!"


if __name__ == "__main__":
        hwg = HellowWorldGTK()
        gtk.main()
```

6 Ukázky zdrojových kódů

Ukázky tvorby modulů v jednotlivých programech jsou demonstrovány na zdrojovém kodu z jazyka C popsaného níže.

6.1 Zdrojový kód v jazyku C

```
/* file example.h */
int add(int a, int b);

/* file example.c */
int add(int a,int b){
  return a + b;
}
```

6.2 SIP

```
/* sip file */

%CModule Example 0

%TypeHeaderCode
#include <example.h>
%End

int add(int a, int b);
/* end of sip file */

\$ sip -c . example.sip

/* configure.py file */
import os
```

```
import sipconfig

build_file = "example.sbf"
config = sipconfig.Configuration()
os.system(" ".join([config.sip_bin, "-c", ".", "-b", build_file,
 "example.sip"]))
makefile = sipconfig.SIPModuleMakefile(config, build_file)
makefile.extra_libs = ["example"]
makefile.generate()
/* end of configure.py */

\$ make
\$ make install

>>> print add(3,3)
>>> 6
```

6.3 BOOST::PYTHON

```
/* boost file */

#include <boost/python.hpp>

BOOST_PYTHON_MODULE(add_ext)
{
    using namespace boost::python;
    def("add", add);
}
/* end of boost file */

\$ make
```

```
\$ make install

>>> print example.add(3,3)
>>> 6
```

6.4 PyInline

```
 /* file example.py */

import PyInline, __main__

m = PyInline.build(code="""
  int add(int a, int b) {
    return a + b;
  }""",
  targetmodule=__main__, language="C")

>>> print example.add(4, 6)
>>> 10
```

6.5 Pyrex

```
/* file example.pyx */

cdef extern from "example.h":
  ctypedef int a
  ctypedef int b
  extern int add
/* end of example.pyx */

\$ pyrexc example.pyx
```

```
\$ gcc -c -fPIC -I/usr/include/python2.7/ example.c

>>> print example.add(1,1)
>>> 2
```

6.6 Ctypes

Není zapotřebí nic komplilovat ani přidávat, stačí využít již zkompilovaný program napsaný v C a pomocí gcc vytvořit knihovnu.

```
>>> from ctypes import *
>>> import os
>>> example = cdll.LoadLibrary(os.getcwd() + '/example.so')
>>> print example.add(2, 2)
>>> 4
```

6.7 Cython

```
/* cython file example.pyx */
cdef extern from "example.h":
    int add(int,int)
/* end of example.pyx */

/* setup.py */
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("example", ["example.pyx"])]

setup(
  name = 'example',
```

```
  cmdclass = {'build_ext': build_ext},
  ext_modules = ext_modules
)
/* end of setup.py */

>>> print example.add(4,4)
>>> 8
```

6.8 SWIG

```
 /* example.i */

 %module example
 %{
 extern int add(int a,int b);
 %}

 extern int add(int a, int b);
/* end of example.i */

\$ swig -python example.i
\$ gcc -c example.c example_wrap.c \
        -I/usr/local/include/python2.7
\$ ld -shared example.o example_wrap.o -o _example.so

>>> import example
>>> example.add(2,2)
>>> 4
```

7 Dokumentace k Python modulu verse

V příloze je uvedená dokumentace k funkcím, ke kterým může programátor přímo přistupovat. Kompletní výpis funkcí lze nalézt ve vygenerované dokumentaci, která je přiložena na CD.

verse_send_connect_request

This function tries to connect to verse server

```
param[in]    *hostname     The string with hostname of the server
param[in]    *service      The string with port number of the server
param[in]    flags         The flags with options of connection
param[out]   *session_id   There will be stored ID of session with verse server

This function will return VC_SUCCESS (0), when the client was able
to start connection to verse server.
```

verse_send_user_authenticate

This function tries to send username and some authentication data (usually password) to the verse server

This function should be called, when client receive User_Authenticate command and callback function registered with py_register_receive_user_authenticate() is called.

```
param[in]   session_id     The ID of session with verse server
param[in]   *username      The string of username
param[in]   auth_type      The authentication method
param[in]   data_length    The length of authentication data in bytes
param[in]   *data          The pointer at authentication data

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

verse_send_connect_terminate

This function switch connection to CLOSING state (start teardown, exit thread) and close connection to verse server.

```
param[in]   session_id     The ID of session with verse server

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

verse_callback_update

This function calls appropriate callback functions, when some system or node commands are in incoming queue

```
param[in]   session_id     The ID of session with verse server

This function returns always VC_SUCCESS.
```

verse_set_debug_level

This function can set debug level of verse client.

```
param[in]   debug_level     This parameter can have values defined in verse.h
```

verse_strerror

Return error message for error_num returned by Verse API functions.

This function should return some string with detail description of error, but it doesn't do anything now.

```
param[in]   error_num     The identifier of error
```

py_register_receive_connect_accept

This function register callback function for command Connect_Accept.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   user_id     The ID of user account
param[in]   avatar_id   The ID of avatar and avatar node
```

py_register_receive_connect_terminate

This function register callback function for situation, when connection to verse server is closed or lost.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   error_num   The error code
```

verse_send_fps

This function tries to negotiate new FPS with server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of the command
param[in]   fps         The number of fps
```

verse_send_node_create

This function tries to send Node_Create command to the server. This function does not send this command directly, but this function add Node_Create to the sending queue. Sending of this command could be delayed due to congestion control or flow control. When client sends this command, then this command is sent with special values: node_id==-1, parent_id==avatar_id and user has to be equal of current user. Note: calling of this function does not guarantee delivery of this command, because connection to the server could be

lost before command Node_Create is sent to the server. The command is added to the priority queue.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of the command

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

py_register_receive_node_create

This function register callback function for command Node_Create.

```
param[in]   (*func)     The pointer at callback function
param[in]   node_id     The ID of new created node
param[in]   parent_id   The ID of parent of new created node
param[in]   user_id     The ID of the owner of new created node
```

verse_send_node_destroy

This function tries to send command Node_Destroy to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of the command
param[in]   node_id     The ID of node that will be destroyed

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

py_register_receive_node_destroy

This function register callback function for command Node_Destroy.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of destroyed node
```

verse_send_node_subscribe

This function tries to send command Node_Subscribe to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID that client wants to be subscribed for
param[in]   level       The level of recursive subscription

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

verse_send_node_unsubscribe

This function tries to send command Node_Unsubscribe to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node that client wants to unsubscribe

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

verse_send_node_link

This function tries to send command Node_Link to the server.

```
param[in]   session_id     The ID of session with verse server
param[in]   prio           The priority of command
param[in]   parent_node_id The ID of parent node
param[in]   child_node_id  The ID of child node

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

py_register_receive_node_link

This function register callback function for command Node_Link.

```
param[in]   (*func)        The pointer at callback function
param[in]   session_id     The ID of session with verse server
param[in]   parent_node_id The ID of parent node
param[in]   child_node_id  The ID of child node
```

verse_send_node_prio

This function tries to send command Node_Priority to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node, where client wants to change priority
param[in]   node_prio   The new priority of Node

This function returns VC_SUCCESS (0), when the session_id
was valid value, it returns VC_FAILURE (1) otherwise.
```

verse_send_taggroup_create

This function send command TagGroup_Create to the server

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node, where taggroup was created
param[in]   *name       The name of new taggroup
```

verse_send_taggroup_destroy

This function send command TagGroup_Destroy to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where taggroup was destroyed
param[in]   taggroup_id The ID of deleted taggroup
```

py_register_receive_taggroup_destroy

This function register callback function for command TagGroup_Destroy.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where taggroup was destroyed
param[in]   taggroup_id The ID of deleted taggroup
```

verse_send_taggroup_subscribe

This function send command TagGroup_Subscribe to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node, where taggroup was subscribed
param[in]   taggroup_id The ID of subscribed taggroup
```

verse_send_taggroup_unsubscribe

This function send command TagGroup_Unsubscribe to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node, where taggroup was unsubscribed
param[in]   taggroup_id The ID of unsubscribed taggroup
```

This function send command Tag_Create to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where tag was created
param[in]   taggroup_id The ID of taggroup, where tag was created
param[in]   tag_id      The ID of new tag
param[in]   type        The type of new tag
param[in]   *name       The name of new tag
```

py_register_receive_tag_create

This function register callback function for command Tag_Create.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where tag was created
param[in]   taggroup_id The ID of taggroup, where tag was created
param[in]   tag_id      The ID of new tag
param[in]   type        The type of new tag
param[in]   *name       The name of new tag
```

verse_send_tag_destroy

This function send command Tag_Destroy to the server.

```
param[in]   session_id   The ID of session with verse server
param[in]   prio         The priority of command
param[in]   node_id      The ID of node, where tag was destroyed
param[in]   taggroup_id  The ID of taggroup, where tag was destroyed
param[in]   tag_id       The ID of destroyed tag
```

py_register_receive_tag_destroy

This function register callback function for command Tag_Destroy.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   prio        The priority of command
param[in]   node_id     The ID of node, where tag was destroyed
param[in]   taggroup_id The ID of taggroup, where tag was destroyed
param[in]   tag_id      The ID of destroyed tag
```

verse_send_tag_set_int8

This function send command Tag_Set_Int8 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int8 value of tag was set
param[in]   tag_id      The ID of int8 tag
param[in]   value       The int8 value
```

py_register_receive_tag_set_int8

This function register callback function for command Tag_Set_Int8.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int8 value of tag was set
param[in]   tag_id      The ID of int8 tag
param[in]   value       The int8 value
```

verse_send_tag_set_uint8

This function send command Tag_Set_Uint8 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint8 value of tag was set
param[in]   tag_id      The ID of uint8 tag
param[in]   value       The uint8 value
```

py_register_receive_tag_set_uint8

This function register callback function for command Tag_Set_Uint8.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint8 value of tag was set
param[in]   tag_id      The ID of uint8 tag
param[in]   value       The uint8 value
```

verse_send_tag_set_int16

This function send command Tag_Set_Int16 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int16 value of tag was set
```

```
param[in]   taggroup_id The ID of taggroup, where int16 value of tag was set
param[in]   tag_id      The ID of int16 tag
param[in]   value       The int16 value
```

py_register_receive_tag_set_int16

This function register callback function for command Tag_Set_Int16.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int16 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int16 value of tag was set
param[in]   tag_id      The ID of int16 tag
param[in]   value       The int16 value
```

verse_send_tag_set_uint16

This function send command Tag_Set_Uint16 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint16 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint16 value of tag was set
param[in]   tag_id      The ID of uint16 tag
param[in]   value       The uint16 value
```

py_register_receive_tag_set_uint16

This function register callback function for command Tag_Set_Uint16.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint16 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint16 value of tag was set
```

```
param[in]   tag_id      The ID of uint16 tag
param[in]   value       The uint16 value
```

verse_send_tag_set_int32

This function send command Tag_Set_Int32 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int32 value of tag was set
param[in]   tag_id      The ID of int32 tag
param[in]   value       The int32 value
```

py_register_receive_tag_set_int32

This function register callback function for command Tag_Set_Int32.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int32 value of tag was set
param[in]   tag_id      The ID of int32 tag
param[in]   value       The int32 value
```

verse_send_tag_set_uint32

This function send command Tag_Set_Uint32 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint32 value of tag was set
param[in]   tag_id      The ID of uint32 tag
param[in]   value       The uint32 value
```

py_register_receive_tag_set_uint32

This function register callback function for command Tag_Set_Uint32.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint32 value of tag was set
param[in]   tag_id      The ID of uint32 tag
param[in]   value       The uint32 value
```

verse_send_tag_set_int64

This function send command Tag_Set_Int64 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int64 value of tag was set
param[in]   tag_id      The ID of int64 tag
param[in]   value       The int64 value
```

py_register_receive_tag_set_int64

This function register callback function for command Tag_Set_Int64.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where int64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where int64 value of tag was set
param[in]   tag_id      The ID of int64 tag
param[in]   value       The int64 value
```

verse_send_tag_set_uint64

This function send command Tag_Set_Uint64 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint64 value of tag was set
param[in]   tag_id      The ID of uint64 tag
param[in]   value       The uint64 value
```

py_register_receive_tag_set_uint64

This function register callback function for command Tag_Set_Uint64.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where uint64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where uint64 value of tag was set
param[in]   tag_id      The ID of uint64 tag
param[in]   value       The uint64 value
```

verse_send_tag_set_real32

This function send command Tag_Set_Real32 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where real32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where real32 value of tag was set
param[in]   tag_id      The ID of real32 tag
param[in]   value       The real32 value
```

py_register_receive_tag_set_real32

This function register callback function for command Tag_Set_Real32.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
```

```
param[in]   node_id     The ID of node, where real32 value of tag was set
param[in]   taggroup_id The ID of taggroup, where real32 value of tag was set
param[in]   tag_id      The ID of real32 tag
param[in]   value       The real32 value
```

verse_send_tag_set_real64

This function send command Tag_Set_Real64 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where real64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where real64 value of tag was set
param[in]   tag_id      The ID of real64 tag
param[in]   value       The real64 value
```

py_register_receive_tag_set_real64

This function register callback function for command Tag_Set_Real64.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where real64 value of tag was set
param[in]   taggroup_id The ID of taggroup, where real64 value of tag was set
param[in]   tag_id      The ID of real64 tag
param[in]   value       The real64 value
```

verse_send_tag_set_string8

This function send command Tag_Set_String8 to the server.

```
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where string8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where string8 value of tag was set
```

```
param[in]   tag_id      The ID of string8 tag
param[in]   value       The string8 value
```

py_register_receive_tag_set_string8

This function register callback function for command Tag_Set_String8.

```
param[in]   (*func)     The pointer at callback function
param[in]   session_id  The ID of session with verse server
param[in]   node_id     The ID of node, where string8 value of tag was set
param[in]   taggroup_id The ID of taggroup, where string8 value of tag was set
param[in]   tag_id      The ID of string8 tag
param[in]   value       The string8 value
```